Data Wrangling

Nate Wells

Math 141, 2/10/21

Outline

In this lecture, we will. . .

Outline

In this lecture, we will...

- Discuss data wrangling and survey the `dplyr` verbs
- Practice decomposing data using the "grammar of wrangling"

Section 1

Data Wrangling

What is Data "Wrangling"?

- Wild data often arrives to us messy—BIG, unsorted, redundant, possibly with data entry/parsing errors.

What is Data "Wrangling"?

- Wild data often arrives to us messy—BIG, unsorted, redundant, possibly with data entry/parsing errors.
- Wrangling is a catch-all term for the process of preparing, manipulating, sorting, relabeling data so it is fit for statistical consumption.

What is Data "Wrangling"?

- Wild data often arrives to us messy—BIG, unsorted, redundant, possibly with data entry/parsing errors.
- Wrangling is a catch-all term for the process of preparing, manipulating, sorting, relabeling data so it is fit for statistical consumption.
- In addition to tidying a data set, data wrangling also allows us to explore components of the data.

What is Data "Wrangling"?

- Wild data often arrives to us messy—BIG, unsorted, redundant, possibly with data entry/parsing errors.
- Wrangling is a catch-all term for the process of preparing, manipulating, sorting, relabeling data so it is fit for statistical consumption.
- In addition to tidying a data set, data wrangling also allows us to explore components of the data.
- Data analysts and survey statisticians spend about $50 - 80\%$ of work-time on data wrangling.

What is Data "Wrangling"?

- Wild data often arrives to us messy—BIG, unsorted, redundant, possibly with data entry/parsing errors.
- Wrangling is a catch-all term for the process of preparing, manipulating, sorting, relabeling data so it is fit for statistical consumption.
- In addition to tidying a data set, data wrangling also allows us to explore components of the data.
- Data analysts and survey statisticians spend about $50 - 80\%$ of work-time on data wrangling.
- As such, it is important to have *consistent* and *efficient* tools for the job.

The dplyr Package

- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:

The dplyr Package

- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:
  1. `filter`
  2. `summarize`
  3. `group_by`
  4. `mutate`
  5. `arrange`
  6. `select`

The dplyr Package

- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:
  1. `filter`
  2. `summarize`
  3. `group_by`
  4. `mutate`
  5. `arrange`
  6. `select`
- Each verb takes a data frame and returns a data frame

The dplyr Package

- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:
  1. `filter`
  2. `summarize`
  3. `group_by`
  4. `mutate`
  5. `arrange`
  6. `select`
- Each verb takes a data frame and returns a data frame
- Verbs can be chained together using a special operator %>% to perform complicated manipulations.

The dplyr Package

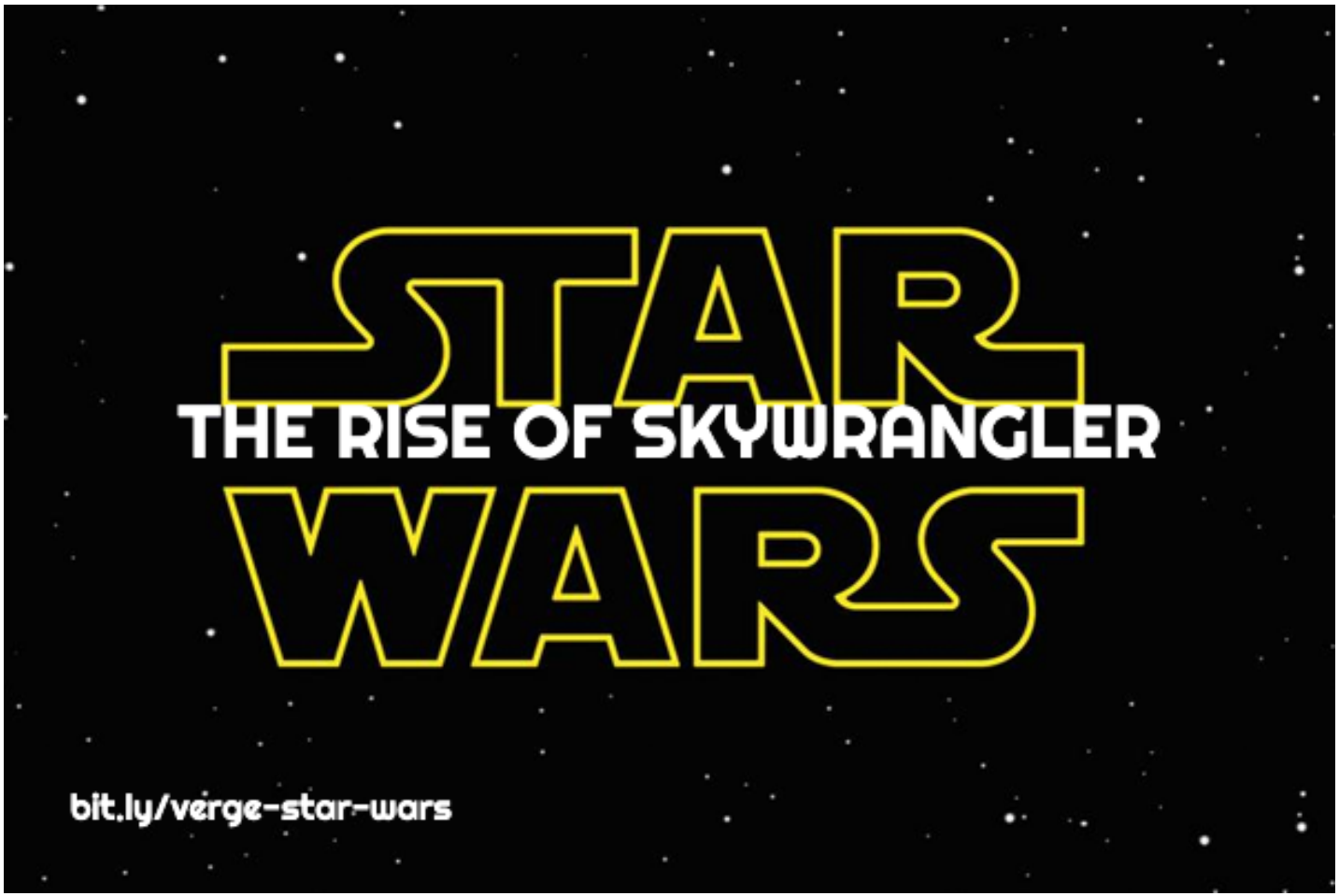
- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:
  1. `filter`
  2. `summarize`
  3. `group_by`
  4. `mutate`
  5. `arrange`
  6. `select`
- Each verb takes a data frame and returns a data frame
- Verbs can be chained together using a special operator %>% to perform complicated manipulations.
- These verbs form a "grammar" of Data Manipulation.

The dplyr Package

- For *tidy* data frames, most wrangling can be performed by 6 `dplyr` functions:
  1. `filter`
  2. `summarize`
  3. `group_by`
  4. `mutate`
  5. `arrange`
  6. `select`
- Each verb takes a data frame and returns a data frame
- Verbs can be chained together using a special operator %>% to perform complicated manipulations.
- These verbs form a "grammar" of Data Manipulation.
  - So even if you aren't using R, they represent the basic components you would think about when manipulating data.

A long time ago, in a galaxy far, far away...

A long time ago, in a galaxy far, far away...



Star Wars: The Rise of Skywrangler

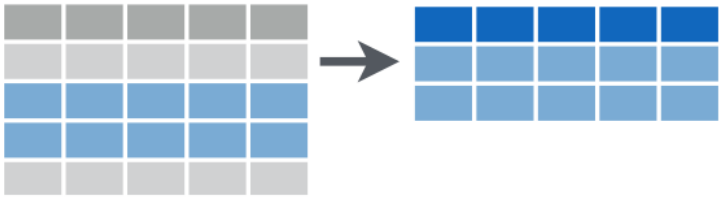
We'll investigate the `starwars` data set from the `dplyr` package

```
head(starwars)
```

```
## # A tibble: 6 x 14
##   name  height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr> <chr>
## 1 Luke~    172    77 blond      fair       blue             19   male  mascu~
## 2 C-3PO    167    75 <NA>       gold       yellow          112   none  mascu~
## 3 R2-D2     96    32 <NA>       white, bl~ red              33   none  mascu~
## 4 Dart~    202   136 none       white      yellow           41.9 male  mascu~
## 5 Leia~    150    49 brown      light      brown            19   fema~ femin~
## 6 Owen~    178   120 brown, gr~ light      blue             52   male  mascu~
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

filter()

Subset Observations (Rows)



filter()


```
filter(starwars, height < 100)
```

```
## # A tibble: 7 x 14
##   name  height  mass hair_color skin_color eye_color birth_year sex   gender
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr> <chr>
## 1 R2-D2     96    32 <NA>       white, bl~ red                33 none  mascu~
## 2 R5-D4     97    32 <NA>       white, red red                NA none  mascu~
## 3 Yoda      66    17 white      green      brown             896 male  mascu~
## 4 Wick~     88    20 brown      brown      brown               8 male  mascu~
## 5 Dud ~     94    45 none       blue, grey yellow             NA male  mascu~
## 6 Ratt~     79    15 none       grey, blue unknown            NA male  mascu~
## 7 R4-P~     96    NA none       silver, r~ red, blue          NA none  femin~
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

select()

Subset Variables (Columns)

select()

Subset Variables (Columns)

```
select(starwars, name, height, mass,homeworld)
```

```
## # A tibble: 87 x 4
##    name               height  mass homeworld
##    <chr>               <int> <dbl> <chr>
##  1 Luke Skywalker        172    77 Tatooine
##  2 C-3PO                 167    75 Tatooine
##  3 R2-D2                  96    32 Naboo
##  4 Darth Vader           202   136 Tatooine
##  5 Leia Organa           150    49 Alderaan
##  6 Owen Lars             178   120 Tatooine
##  7 Beru Whitesun lars    165    75 Tatooine
##  8 R5-D4                  97    32 Tatooine
##  9 Biggs Darklighter     183    84 Tatooine
## 10 Obi-Wan Kenobi        182    77 Stewjon
## #    with 77 more rows
```

summarize()

summarize()


```
summarize(starwars,
          Avg_Height = mean(height, na.rm = T),
          Median_Height = median(height, na.rm = T))
```

```
## # A tibble: 1 x 2
##   Avg_Height Median_Height
##        <dbl>         <int>
## 1       174.           180
```

group_by()

Link data according to levels of a variable. Usually followed by `summarize()`

group_by()

Link data according to levels of a variable. Usually followed by `summarize()`

```
grouped_sw <- group_by(starwars, gender)
summarize(grouped_sw, Avg_Height = mean(height, na.rm = T))
```

```
## # A tibble: 3 x 2
##   gender    Avg_Height
##   <chr>          <dbl>
## 1 feminine        165.
## 2 masculine       177.
## 3 <NA>            181.
```

mutate()

mutate()

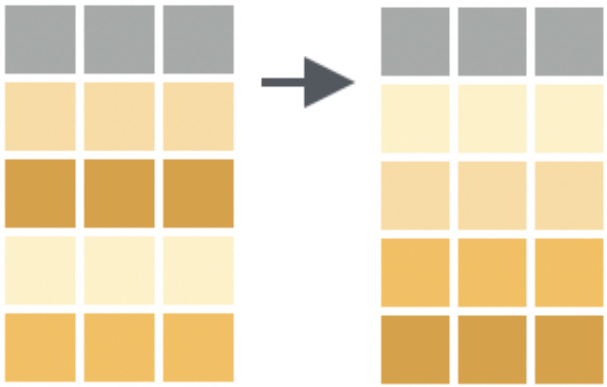
Make New Variables

```
mutated_sw <- mutate(starwars, height_ft = height/30.48)
select(mutated_sw, name, height_ft, everything())
```

```
## # A tibble: 87 x 15
##    name  height_ft height  mass hair_color skin_color eye_color birth_year sex
##    <chr>     <dbl>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
##  1 Luke~      5.64    172    77 blond      fair       blue            19   male
##  2 C-3PO      5.48    167    75 <NA>       gold       yellow         112   none
##  3 R2-D2      3.15     96    32 <NA>       white, bl~ red             33   none
##  4 Dart~      6.63    202   136 none       white      yellow          41.9 male
##  5 Leia~      4.92    150    49 brown      light      brown           19   fema~
##  6 Owen~      5.84    178   120 brown, gr~ light      blue            52   male
##  7 Beru~      5.41    165    75 brown      light      blue            47   fema~
##  8 R5-D4      3.18     97    32 <NA>       white, red red             NA   none
##  9 Bigg~      6.00    183    84 black      light      brown           24   male
## 10 Obi-~      5.97    182    77 auburn, w~ fair       blue-gray       57   male
## # ... with 77 more rows, and 6 more variables: gender <chr>, homeworld <chr>,
```

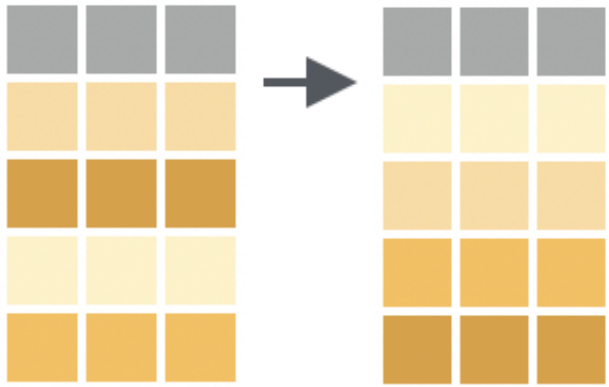
arrange()

Sort the rows



arrange()

Sort the rows



```
arrange(starwars,mass)
```

```
## # A tibble: 87 x 14
##    name  height  mass hair_color skin_color eye_color birth_year sex   gender
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr> <chr>
##  1 Ratt~     79    15 none       grey, blue unknown            NA male  mascu~
##  2 Yoda      66    17 white      green      brown             896 male  mascu~
##  3 Wick~     88    20 brown      brown      brown               8 male  mascu~
##  4 R2-D2     96    32 <NA>       white, bl~ red                33 none  mascu~
##  5 R5-D4     97    32 <NA>       white, red red                NA none  mascu~
##  6 Sebu~    112    40 none       grey, red  orange             NA male  mascu~
##  7 Dud ~     94    45 none       blue, grey yellow             NA male  mascu~
```

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together
- Suppose you want to perform a sequence of operations on a data frame `df` with several variables:

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together
- Suppose you want to perform a sequence of operations on a data frame `df` with several variables:
  1. `selecting` only the first variable with the function `select()`
  2. `filtering` observations in a certain range with the function `filter()`
  3. `arranging` observations in increasing order with the function `arrange()`

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together
- Suppose you want to perform a sequence of operations on a data frame `df` with several variables:
  1. `selecting` only the first variable with the function `select()`
  2. `filtering` observations in a certain range with the function `filter()`
  3. `arranging` observations in increasing order with the function `arrange()`
- One way to code this (ignoring extra arguments) is:

```
arrange(filter(select (my_data)))
```

- This method has two primary problems:

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together
- Suppose you want to perform a sequence of operations on a data frame `df` with several variables:
  1. `selecting` only the first variable with the function `select()`
  2. `filtering` observations in a certain range with the function `filter()`
  3. `arranging` observations in increasing order with the function `arrange()`
- One way to code this (ignoring extra arguments) is:

```
arrange(filter(select (my_data)))
```

- This method has two primary problems:
  1. Code quickly become overwhelming to read and review (especially as number of functions and arguments increases)

The Pipe

- The pipe operator %>% (read as "pipe" or "then") chains verbs together
- Suppose you want to perform a sequence of operations on a data frame `df` with several variables:
  1. `selecting` only the first variable with the function `select()`
  2. `filtering` observations in a certain range with the function `filter()`
  3. `arranging` observations in increasing order with the function `arrange()`
- One way to code this (ignoring extra arguments) is:

```
arrange(filter(select (my_data)))
```

- This method has two primary problems:
  1. Code quickly become overwhelming to read and review (especially as number of functions and arguments increases)
  2. The operations (as read from left to right) appear in the *opposite* order to how they are performed

Pipe Composition

- Instead, we can obtain the same output using the pipe:

Pipe Composition

- Instead, we can obtain the same output using the pipe:

```
df %>%
  select() %>%
  filter() %>%
  arrange()
```

Pipe Composition

- Instead, we can obtain the same output using the pipe:

```
df %>%
  select() %>%
  filter() %>%
  arrange()
```

- Reading %>% as "then", this sequence translates to

Pipe Composition

- Instead, we can obtain the same output using the pipe:

```
df %>%
  select() %>%
  filter() %>%
  arrange()
```

- Reading %>% as "then", this sequence translates to
  1. Take `df` *then*
  2. Use this output as input of `select()` *then*
  3. Use this output as input of `filter()` *then*
  4. Use this output as input of `arrange()`

Pipe Composition

- Instead, we can obtain the same output using the pipe:

```
df %>%
  select() %>%
  filter() %>%
  arrange()
```

- Reading %>% as "then", this sequence translates to
  1. Take `df` *then*
  2. Use this output as input of `select()` *then*
  3. Use this output as input of `filter()` *then*
  4. Use this output as input of `arrange()`

- Advantages:
  - The pipe sequence is much more readable.
  - Much easier to add more functions to the mix at a later time (since they can be tacked on at the end of the sequence)

Section 2

Data Decomposition

Math 141 Survey

year	historian	alcohol	hogwarts	hot_dog	college_app	dog_pants	social	economic
Sophomore	Herodotus	1.0	Ravenclaw	Maybe	5	Back legs	4	5
Sophomore	Thucydides	1.0	Ravenclaw	No	8	Back legs	3	3
Sophomore	Thucydides	2.0	Gryffindor	No	10	Back legs	5	5
Sophomore	Thucydides	0.0	Hufflepuff	No	11	Back legs	4	4
Sophomore	None	1.0	Slytherin	No	5	Back legs	3	3
Junior	Herodotus	5.0	Slytherin	No	2	Back legs	5	3
Sophomore	Herodotus	0.0	Ravenclaw	No	2	Back legs	3	4
Senior	Herodotus	2.0	Slytherin	No	9	All legs	1	1
Junior	Herodotus	1.0	Hufflepuff	Yes	6	All legs	2	2
Sophomore	Thucydides	3.0	Gryffindor	Yes	1	Back legs	2	4
Junior	Herodotus	0.0	Ravenclaw	No	3	All legs	6	7
Sophomore	Herodotus	1.0	Gryffindor	Yes	7	All legs	1	6
Sophomore	Herodotus	3.0	Ravenclaw	Yes	20	All legs	4	5
Sophomore	Herodotus	4.0	Gryffindor	No	16	All legs	3	2
Junior	Herodotus	0.0	Ravenclaw	No	3	Back legs	2	2
Freshman	Thucydides	0.0	NA	Yes	10	All legs	5	6
Junior	Thucydides	0.1	Gryffindor	No	12	Back legs	3	1
Sophomore	Thucydides	0.5	Slytherin	NA	1	NA	3	8
Freshman	Herodotus	2.0	Gryffindor	Yes	3	Back legs	5	7
Sophomore	Herodotus	0.0	Slytherin	No	13	Back legs	3	4

Section 3

Summarizing with dplyr

The dplyr package

- The dplyr (dee-plier) package provides a set of specialized tools for manipulating dataframes.

The dplyr package

- The dplyr (dee-plier) package provides a set of specialized tools for manipulating dataframes.
- While dplyr contains many functions (we'll see at least 6 over the next few days), for now we focus on just one: summarize (or summarise)

The dplyr package

- The dplyr (dee-plier) package provides a set of specialized tools for manipulating dataframes.
- While dplyr contains many functions (we'll see at least 6 over the next few days), for now we focus on just one: summarize (or summarise)
- Previously, we applied functions like mean(), sd() and quantile() to columns of a data frame to get summary statistics:

The dplyr package

- The dplyr (dee-plier) package provides a set of specialized tools for manipulating dataframes.
- While dplyr contains many functions (we'll see at least 6 over the next few days), for now we focus on just one: summarize (or summarise)
- Previously, we applied functions like mean(), sd() and quantile() to columns of a data frame to get summary statistics:

```
mean(biketown$Distance_Miles)
```

```
## [1] 2.044768
```

The dplyr package

- The dplyr (dee-plier) package provides a set of specialized tools for manipulating dataframes.
- While dplyr contains many functions (we'll see at least 6 over the next few days), for now we focus on just one: summarize (or summarise)
- Previously, we applied functions like mean(), sd() and quantile() to columns of a data frame to get summary statistics:

```
mean(biketown$Distance_Miles)
```

```
## [1] 2.044768
```

- But it would be nice to have an easy way to store multiple summary statistics in a data frame

The summarize function

The summarize function takes a data frame, applies specified summary functions to 1 or more columns, and returns a data frame of the results.

The summarize function

The summarize function takes a data frame, applies specified summary functions to 1 or more columns, and returns a data frame of the results.

```
library(dplyr)
summarize(
  biketown,
    Mean_Distance = mean(Distance_Miles),
    SD_Distance = sd(Distance_Miles),
    Median_StartHour = median(StartHour),
    IQR_StartHour = IQR(StartHour)
)
```

```
## # A tibble: 1 x 4
##   Mean_Distance SD_Distance Median_StartHour IQR_StartHour
##           <dbl>       <dbl>             <int>         <dbl>
## 1          2.04        1.95                15             7
```

- Note that code is separated by line breaks for improved readability
- New column names can be arbitrary (but it's nice if they are informative)

The summarize function

The summarize function takes a data frame, applies specified summary functions to 1 or more columns, and returns a data frame of the results.

```
library(dplyr)
summarize(
  biketown,
    These = mean(Distance_Miles),
    Can = sd(Distance_Miles),
    Be = median(StartHour),
    Whatever = IQR(StartHour)
)
```

```
## # A tibble: 1 x 4
##   These   Can    Be Whatever
##   <dbl> <dbl> <int>    <dbl>
## 1  2.04  1.95    15        7
```

- Note that code is separated by line breaks for improved readability
- New column names can be arbitrary (but it's nice if they are informative)

Extending summarize

- The summarize function can be combined with many common R functions that take a list of values and return a single value:

Extending summarize

- The summarize function can be combined with many common R functions that take a list of values and return a single value:
  - mean()
  - sd()
  - median()
  - IQR()
  - quantile()
  - sum()
  - min()
  - max()
  - n()

Extending summarize

- The summarize function can be combined with many common R functions that take a list of values and return a single value:
  - mean()
  - sd()
  - median()
  - IQR()
  - quantile()
  - sum()
  - min()
  - max()
  - n()
- It's helpful to save the summarize dataframe for later access:

Extending summarize

- The summarize function can be combined with many common R functions that take a list of values and return a single value:
  - mean()
  - sd()
  - median()
  - IQR()
  - quantile()
  - sum()
  - min()
  - max()
  - n()
- It's helpful to save the summarize dataframe for later access:

```
distance_summary <- summarise(biketown,
                              mean_dist = mean(Distance_Miles),
                              sd_dist = sd(Distance_Miles))
```

Extending summarize

- The summarize function can be combined with many common R functions that take a list of values and return a single value:
  - mean()
  - sd()
  - median()
  - IQR()
  - quantile()
  - sum()
  - min()
  - max()
  - n()
- It's helpful to save the summarize dataframe for later access:

```
distance_summary <- summarise(biketown,
                              mean_dist = mean(Distance_Miles),
                              sd_dist = sd(Distance_Miles))
```

```
distance_summary$mean_dist
```

```
## [1] 2.044768
```

```
distance_summary$sd_dist
```

```
## [1] 1.950804
```